AMENDMENTS TO THE SPECIFICATION:

Page 1, between the title and first paragraph, insert the following heading and subheading:

BACKGROUND OF THE INVENTION

1. Field of the Invention

Page 1, above the paragraph beginning at line 7, insert the following subheading:

2. Discussion of Prior art

Page 1, above the paragraph beginning at line 18, insert the following heading:

SUMMARY OF THE INVENTION

Page 3, above the paragraph beginning at line 16, insert the following heading:

BRIEF DESCRIPTION OF THE DRAWINGS

Page 4, above the paragraph beginning at line 21, insert the following heading:

DETAILED DISCUSSION OF EMBODIMENTS

Page 11, amend the paragraph beginning at line 22 and continuing to page 12, line 5 as follows:

Figure 9 schematically illustrates a register bank 96. This register bank is based upon the ARM processor programmer's model for user mode operation in accordance with processors designed by ARM Limited, Cambridge, ~~England)~~England. In practice, further registers may be provided for other processor modes, but these have been omitted for the sake of clarity. The normal data registers R0 to R15 are provided for holding data values. The registers R13, R14 and R15 typically serve to store the program counter value, the branch return address value and

the stack pointer, which tend to be none security related data values. Accordingly, transition balancing upon data writes is not necessary for R13, R14 and R15. A trash register RT is provided within the register bank 96 for use in association with conditional writes which fail their condition codes thus, a conditional write instruction which fails its condition code would not normally make any write. However, with this system such a failed conditional write instruction nevertheless writes a data value to the trash data register RT even though the condition codes have failed. This masks any difference in power consumption or timing that might be associated with condition code failure or condition code passing of a conditional write operation. The trash data register RT does not appear in the programmer's model in a way that enables it to be addressed with a register specifying operand within an instruction.

Page 13, amend the paragraph beginning at line 25 and continuing to page 14, line 6 as follow:

Figure ~~13~~12 is a flow diagram illustrating the action of the dummy data register RD to provide writes when a write operation fails its condition code(s). At step 118 the control logic waits for an instruction to be received. This control logic may be the instruction decoder 68 or other logic. Step 120 determines whether or not the instruction failed its condition codes. If the instruction does not fail its condition code, then it is normally executed at step 122 and makes its write to the register specified by the register operand within that instruction. If the instruction does fail its condition codes, then processing proceeds to step 124 at which a determination is made as to whether or not dummy data register writes are enabled. If these are not enabled, then processing terminates. If dummy data register writes are enabled, then processing proceeds to step 126 at which a write of the data value calculated by the condition code failed instruction is

written to the trash data register RT even though the condition codes failed. This balances the power consumption and timing irrespective of a condition code pass or a condition code fail. It will be appreciated that the trash data register RT is also subject to the transition balancing mechanisms previously discussed.

Page 14, amend the paragraph beginning at line 28 and continuing to page 15, line 3 as follows:

Figure 14 schematically illustrates the handling of a received Java bytecode. At step 138 a Java bytecode is received. Step 140 determines whether or not the Java decoder 130 is enabled. The pseudo random enabling and disabling of the Java ~~decoder 130 effectively~~ causes a branch (step 144) to either step 142 at which the bytecode is always emulated or an attempt to execute the instruction in hardware at step 146. This obscures/masks the power signature associated with Java bytecode execution. If the determination at step 146 is that the particular Java bytecode concerned is not supported by the Java decoder 130, then this Java bytecode will also be emulated in software at step 142. However, if the Java bytecode is supported in hardware, then it is executed in hardware at step 146.